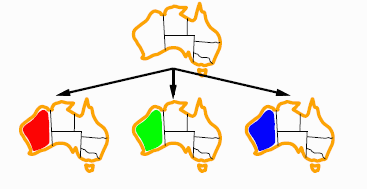
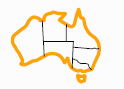
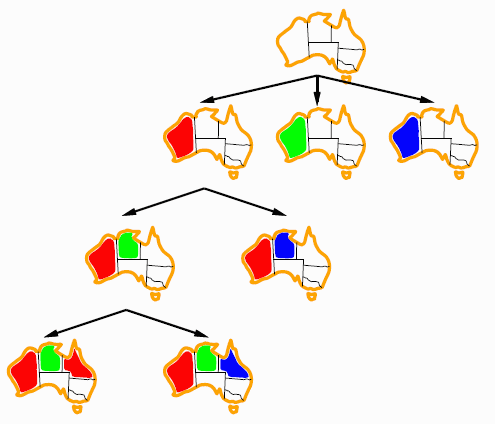
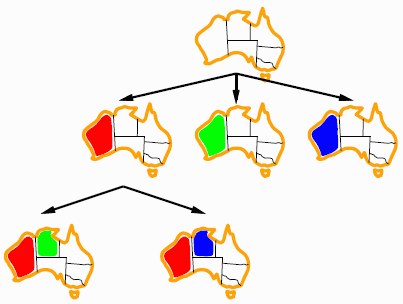


**Backtracking algorithm pseudocode:**

**def BACKTRACKING\_SEARCH(csp) :  
   return RECURSIVE\_BACKTRACKING( {}, csp)  
  
def RECURSIVE\_BACKTRACKING( assignment, csp) :  
   if COMPLETE(assignment, csp.VARIABLES ) : return assignment  
   var = SELECT\_UNASSIGNED\_VARIABLE( csp.VARIABLES, assignment, csp)  
   for value in ORDER\_DOMAIN\_VALUES( var, assignment, csp) :  
      if CONSISTENT(var, value, assignment, csp.CONSTRAINTS, csp.NEIGHBORS) :  
         assignment[var]=value                     # Add var=value  
         result = RECURSIVE\_BACKTRACKING(assignment, csp)  
         if result != 'failure' : return result  
         del assignment[var]                          # Remove var=value  
   return 'failure'**

Backtracking example





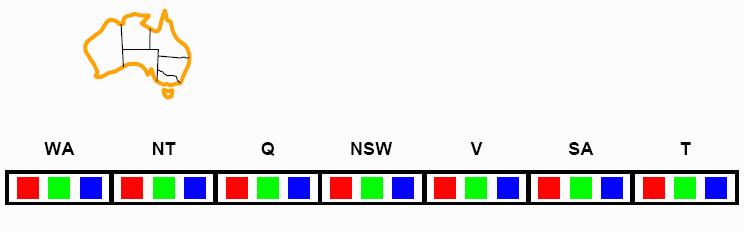
Constraints on SA will eventually cause failure when WA != Q. When not the same color (bottom right), SA cannot be assigned.

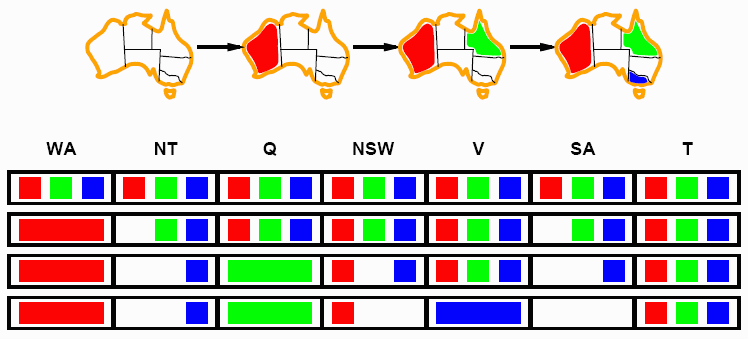
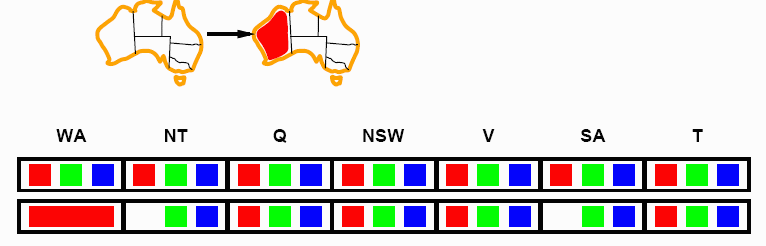
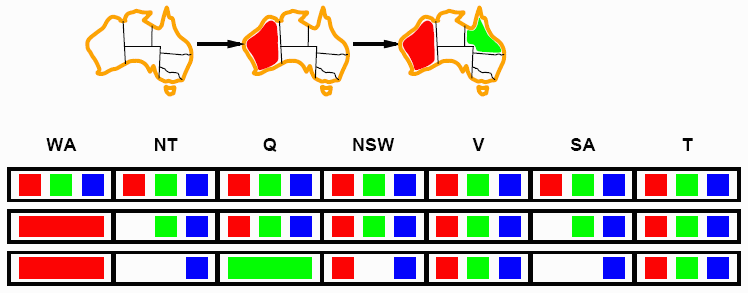
The algorithm will backtrack to a node with unexplored states. For example, such as WA=red, NT=blue.

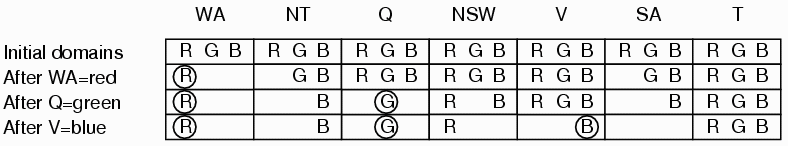
Forward checking

Keep only legal values in domain for *unassigned* variables

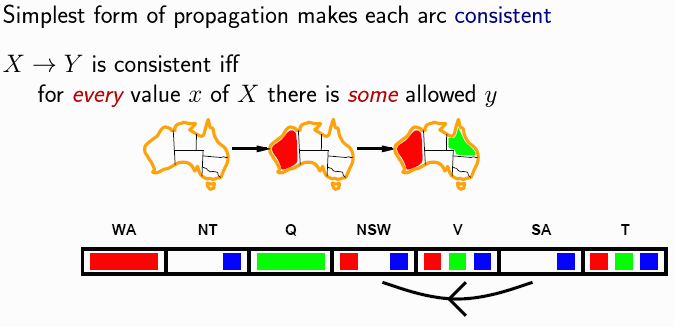
1. Prune illegal values from domain of connected (neighbors) unassigned variables after each assignment
2. Backtrack search when any variable has no legal values
3. When backtracking to undo an assignment, restore pruned values to domain of connected (neighbors) unassigned variables

Reduces search tree by pruning branches destined to eventually fail due to values that violate constraints in the domains of neighbors

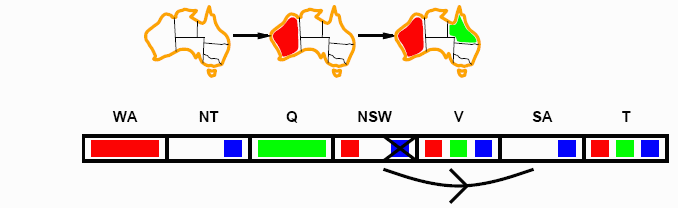




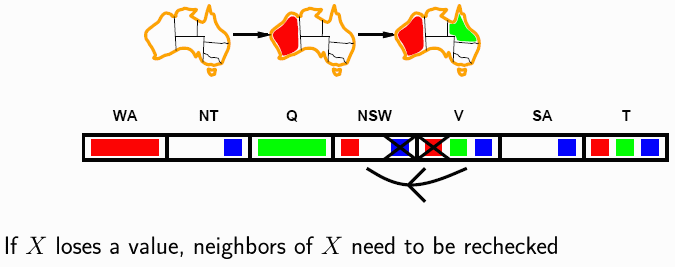
## http://homepages.ius.edu/RWISMAN/C463/html/chapter5-3.gifArc consistency

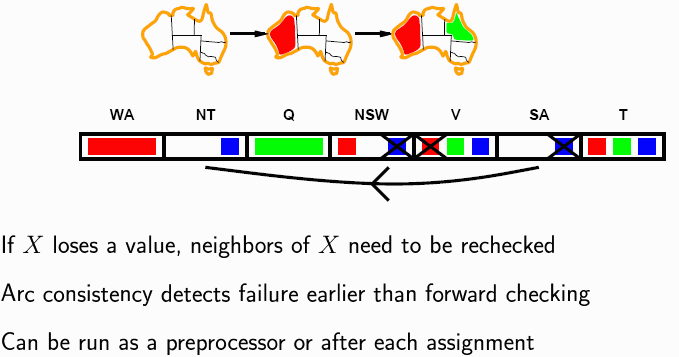


Arc from SA to NSW is consistent because SA (blue) allows NSW (red)



Arc from NSW to SA is not consistent because NSW (blue) does not allow SA (blue) Can be made consistent by removing blue from NSW domain



After removing blue from NSW, must check neighbors for consistency leading to removal of red from V.

After removing blue, SA domain is {}, pruning SA completely for the two assignments {NSW=red, Q=green} and forcing backtracking.

Arc Consistency Algorithm Pseudocode:

|  |
| --- |
| function AC3( *csp* ) returns the CSP possibly with reduced domains    local variables: *queue*, queue of arcs, initially all the arcs in *csp* loop while *queue*  is not empty do        (Xi, Xj) = REMOVE-FRONT(*queue)* if REMOVE-INCONSISTENT(Xi, Xj) then     # Neighbors may be inconsistent           for each Xk in NEIGHBORS[Xi] do          # add to queue to check              add(Xk, Xi) to *queue* |
| function REMOVE-INCONSISTENT(Xi, Xj) returns True iff Xi value removed      *removed* = False     for each x in DOMAIN[Xi] do         if (x, y) is *inconsistent* for every value y in DOMAIN[Xj]              then delete x from DOMAIN[Xi]              removed = True     return removed |

O(n2d3) is worst case time but cannot detect all failures in polynomial time.

AC3 checks all neighbors and removes inconsistencies, the effectiveness can be seen when applied to the Australia map coloring.

* In conjunction with a backtracking search, AC3 eliminates backtracking yielding a linear search tree with no branches.
* Recall that backtracking occurs when all domain values of the assigned variable have been exhausted and no solution was found.
* On failure, the variable and value are removed from the list of assigned variables, necessary before backtracking to a previous node in the search tree.

In practice, AC3 can be called one time before calling the backtracking search or each time after assigning a variable a consistent value according to problem constraints.

* When backtracking occurs, a branch for a value from a different variable is explored.
* Since the effect of AC3 is to remove inconsistent domain values from all connected variables, this prunes search tree branches recognized to eventually lead to failure. Removed values are never explored.
* A side-effect of AC3 is that domains are modified.

Problems

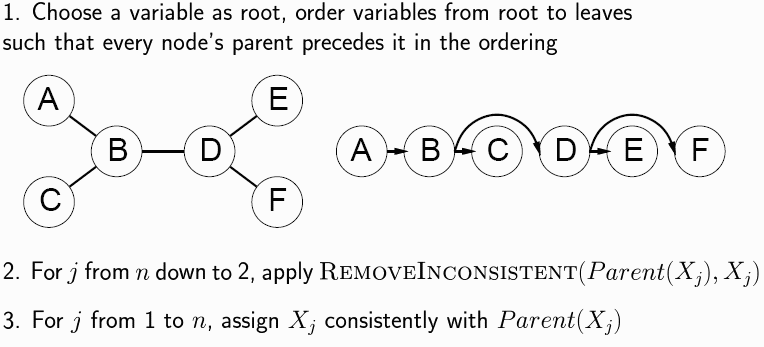
Arc consistency does not reveal every possible inconsistency.

The partial assignment {WA=red, NSW=red} is inconsistent but AC3 will not find it because only considers 2 variables at a time.

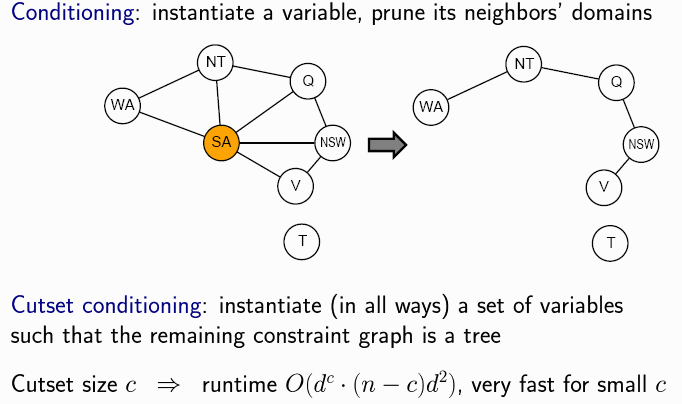
## http://homepages.ius.edu/RWISMAN/C463/html/chapter5-32.gifTree-structured CSPs

t

## Algorithm for tree-structured CSPs



## Nearly tree-structured CSPs



## http://homepages.ius.edu/RWISMAN/C463/html/chapter5-38.gifSummary

MinMax Algorithm:

**function** minimax(node, depth, maximizingPlayer)

**if** depth = 0 **or** node is a terminal node

**return** the heuristic value of node

**if** maximizingPlayer

bestValue := -∞

**for each** child of node

val := minimax(child, depth - 1, FALSE))

bestValue := max(bestValue, val);

**return** bestValue

**else**

bestValue := +∞

**for each** child of node

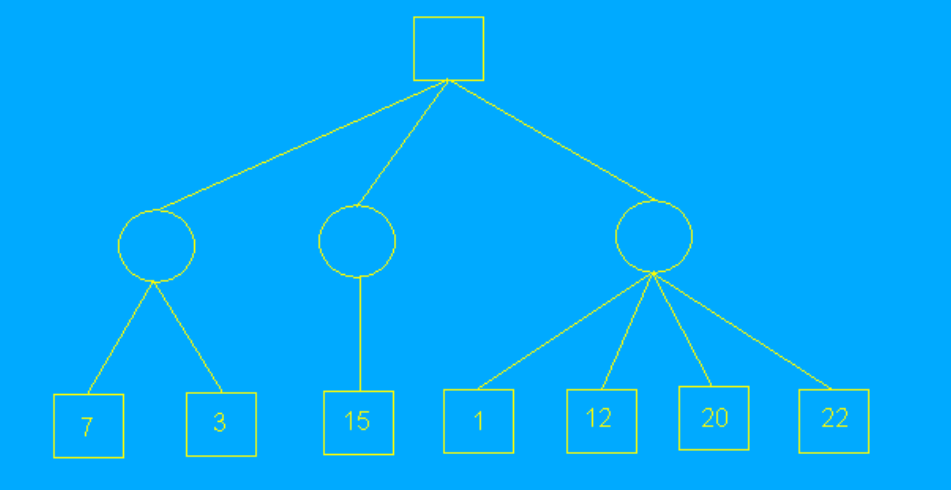
val := minimax(child, depth - 1, TRUE))

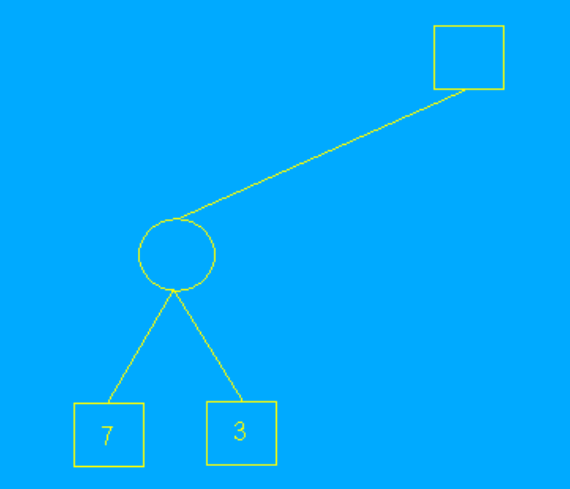
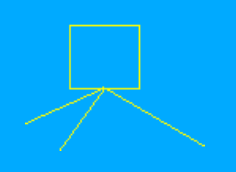
bestValue := min(bestValue, val);

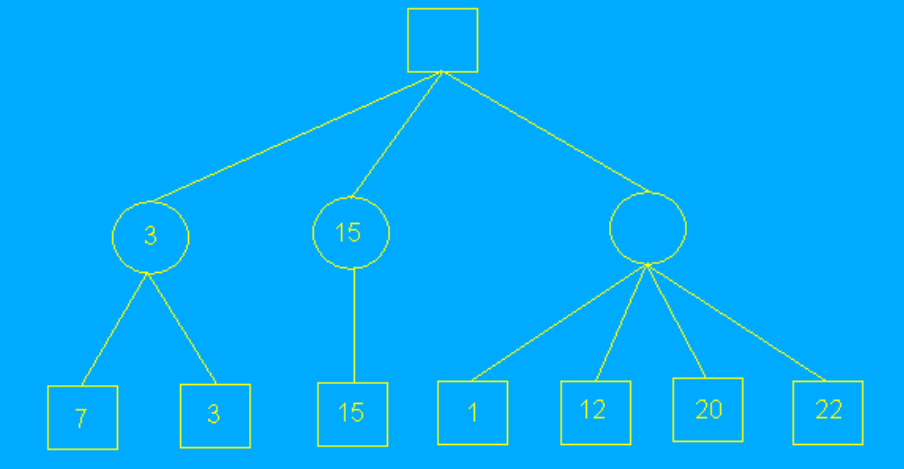
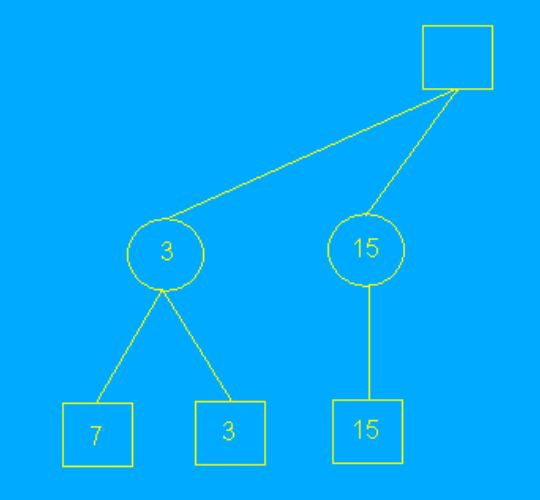
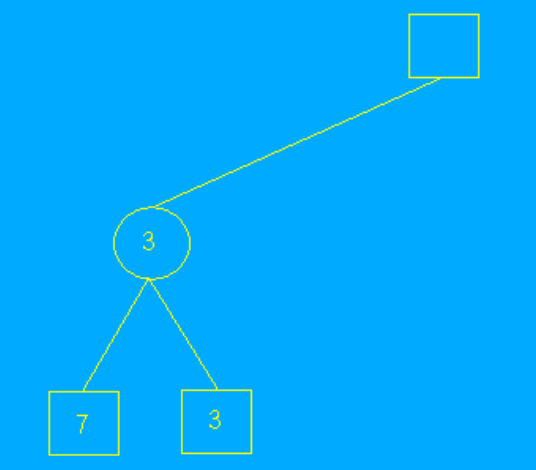
**return** bestValue

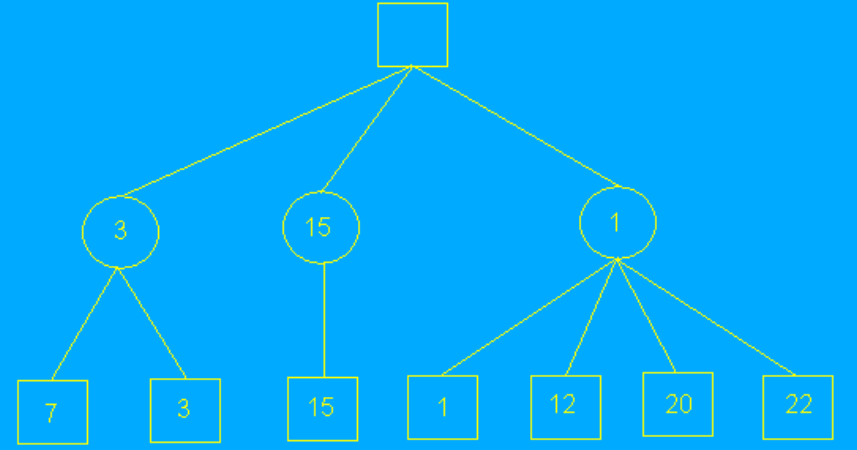
*(\* Initial call for maximizing player \*)*

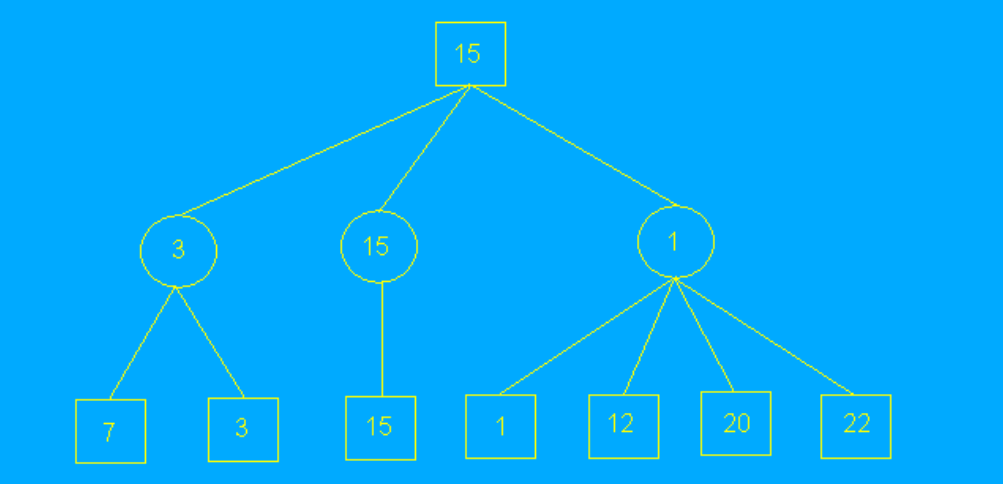
minimax(origin, depth, TRUE)

MinMax:









Alpha Beta Pruning:

**function** alphabeta(node, depth, α, β, maximizingPlayer)

**if** depth = 0 **or** node is a terminal node

**return** the heuristic value of node

**if** maximizingPlayer

**for each** child of node

α := max(α, alphabeta(child, depth - 1, α, β, FALSE))

**if** β ≤ α

**break** *(\* β cut-off \*)*

**return** α

**else**

**for each** child of node

β := min(β, alphabeta(child, depth - 1, α, β, TRUE))

**if** β ≤ α

**break** *(\* α cut-off \*)*

**return** β

*(\* Initial call \*)*

alphabeta(origin, depth, -[∞](http://en.wikipedia.org/wiki/Infinity), +[∞](http://en.wikipedia.org/wiki/Infinity), TRUE)

Alpha beta pruning example:

